Provenance for SPARQL queries

C. V. Damásio[1] and A. Analyti[2] and G. Antoniou[3]

[1] CENTRIA, Departamento de Informática Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal.
cd@fct.unl.pt

[2] Institute of Computer Science, FORTH-ICS, Crete, Greece
analyti@ics.forth.gr

[3] Institute of Computer Science, FORTH-ICS, and
Department of Computer Science, University of Crete, Crete, Greece
antoniou@ics.forth.gr

Abstract. Determining trust of data available in the Semantic Web is fundamental for applications and users, in particular for linked open data obtained from SPARQL endpoints. There exist several proposals in the literature to annotate SPARQL query results with values from abstract models, adapting the seminal works on provenance for annotated relational databases. We provide an approach capable of providing provenance information for a large and significant fragment of SPARQL 1.1, including for the first time the major non-monotonic constructs under multiset semantics. The approach is based on the translation of SPARQL into relational queries over annotated relations with values of the most general m-semiring, and in this way also refuting a claim in the literature that the `OPTIONAL` construct of SPARQL cannot be captured appropriately with the known abstract models.

Keywords: How-provenance, SPARQL queries, m-semirings, difference

Overview

This document presents the proof of the main result of the paper entitled "Provenance for SPARQL queries" to appear in Proceedings of ISWC 2012 Boston, edited by Bernstein, Cudre-Mauroux, Heflin et al., Springer. The original publication will be available at `www.springerlink.com` and this document will be substituted by the authors' version after publication.

A Proof of the main result

The rationale for obtaining how-provenance for SPARQL is to represent each solution mapping as a tuple of a relational algebra query constructed from the original SPARQL graph pattern. The construction is intricate and fully specified, and is inspired from the translation of full SPARQL 1.0 queries into SQL,

as detailed in [6], and into Datalog in [11]. Here, we follow a similar strategy but for simplicity of presentation we assume that a given RDF dataset $D = \{G_0, (<\mathtt{u_1}>, G_1), (<\mathtt{u_2}>, G2), \ldots, (<\mathtt{u_n}>, G_n)\}$ is represented by the two relations: `Graphs(gid,IRI)` and `Quads(gid,sub,pred,obj)`. The former stores information about the graphs in the dataset D where `gid` is a numeric graph identifier, and `IRI` an IRI reference. The relation `Quads` stores the triples of every graph in the RDF dataset. Different implementations may immediately adapt the translation provided here in this section to their own schema.

Relation `Graphs(gid,IRI)` contains a tuple $(i, <\mathtt{u_i}>)$ for each named graph $(<\mathtt{u_i}>, G_i)$, and the tuple $(0, <\ >)$ for the default graph, while relation `Quads(gid,sub,pred,obj)` stores a tuple of the form (i, s, p, o) for each triple $(s, p, o) \in G_i$[4]. With this encoding, the default graph always has identifier 0, and all the graph identifiers are consecutive integers.

It is also assumed the existence of a special value `unb`, distinct from the encoding of any RDF term, to represent that a particular variable is unbound in the solution mapping. This is required in order to be able to represent solution mappings as tuples with fixed and known arity. Moreover, we assume that the variables are totally ordered (e.g. lexicographically). The translation requires the full power of relational algebra, and notice that bag semantics is assumed (duplicates are allowed) in order to obey to the cardinality restrictions of SPARQL algebra operators [1].

Theorem 1 (Correctness of translation). *Given a graph pattern P and a RDF dataset $D(G)$ the process of evaluating the query is performed as follows:*

1. *Construct the base relations* `Graphs` *and* `Quads` *from $D(G)$;*
2. *Evaluate $[SPARQL(P, D(G), V)]_{\mathcal{R}} = \Pi_V \left[\sigma_{G'=0}\left([()]_{\mathcal{R}}^{G'} \bowtie [P]_{\mathcal{R}}^{G'}\right)\right]$ with respect to the base relations* `Graphs` *and* `Quads`*, where G' is a new attribute name and $V \subseteq var(P)$.*

Moreover, the tuples of relational algebra query (2) are in one-to-one correspondence with the solution mappings of $[\![P]\!]_{D(G)}$ when $V = var(P)$, and where an attribute mapped to `unb` *represents that the corresponding variable does not belong to the domain of the solution mapping.*

Proof. The induction proof will construct an expression containing as attributes the graph attribute and an attribute for each in-scope variable of the graph pattern. Assume that the active graph is given and has id j (0 for the default graph, or $1 \le j \le n$ for the case of a named graph). We show that the cardinality of solutions of each SPARQL algebra operator is respected for the active graph, which is the most difficult part. Notice that the graphs by definition do not have duplicate triples, and therefore there will not be duplicates in the original base relations `Graphs` and `Quads`.

[4] For simplicity `sub`, `pred`, and `obj` are text attributes storing lexical forms of the triples' components. We assume that datatype literals have been normalized, and blank nodes are distinct in each graph. The only constraint is that different RDF terms must be represented by different strings; this can be easily guaranteed.

Empty graph pattern Recall that he empty graph pattern is translated into $[()]_{\mathcal{R}}^{G} = \Pi_G \left[\rho_{G \leftarrow \mathtt{gid}}(\mathtt{Graphs})\right]$. This means that the result of the relational algebra query returns as many as tuples as graphs in the RDF dataset, and at least a tuple with identifier 0 for the initial default graph. Moreover, note that for each particular graph id (default or named) this relation has exactly 1 tuple with that id.

Triple pattern Since there are no duplicate triples in the graphs, then the number of possible solutions of a triple pattern are exactly the number of triples that match the pattern, one for each instance. We analyse the correctness of the translation of the triple pattern according to the number of variables that may occur on it: no variables, one variable, two variables and three variables.

0 vars: In this case the triple pattern $t = (t_1, t_2, t_3)$ contains only RDF terms which might be identical, or not. In this case we obtain the relational algebra expression $\Pi_G \left[\rho_{G \leftarrow \mathtt{gid}}\left(\sigma_{\mathtt{sub}=t_1 \wedge \mathtt{pred}=t_2 \wedge \mathtt{obj}=t_3}(\mathtt{Quads})\right)\right]$. The resulting relation has just the attribute (column) G. If the triple occurs in graph i then the resulting relation will contain a tuple having value i in attribute G; otherwise no tuple will occur for graph i.

1 var: We have several cases to consider here: there is just one occurrence of a variable, two or three.
Consider that the variable occurs in the subject of the triple; for the remaining cases in predicate or object the reasoning is similar. So, let $t = (?v, t_2, t_3)$, obtaining the relational algebra expression

$$\Pi_{G,v} \left[\rho_{G \leftarrow \mathtt{gid}, v \leftarrow \mathtt{sub}}\left(\sigma_{\mathtt{pred}=t_2 \wedge \mathtt{obj}=t_3}(\mathtt{Quads})\right)\right]$$

The resulting relation has two columns, one for the graph G and one for collecting the bindings for v (i.e. the solution). If the triple occurs in graph i then the resulting relation will contain a tuple having value i in attribute G; otherwise no tuple will occur for graph i.
Consider now that the variable occurs in the subject and predicate of the triple; for the remaining cases the reasoning is similar. So, let $t = (?v, ?v, t_3)$, obtaining the relational algebra expression

$$\Pi_{G,v} \left[\rho_{G \leftarrow \mathtt{gid}, v \leftarrow \mathtt{sub}}\left(\sigma_{\mathtt{sub}=\mathtt{pred} \wedge \mathtt{obj}=t_3}(\mathtt{Quads})\right)\right]$$

The resulting relation has two columns, one for the graph G and one for collecting the bindings for v (i.e. the solution). The selection condition guarantees that the obtained instances match the triple pattern, obtaining for each graph i one tuple for each possible value of v.
If there are three occurrences of the variable $?v$ then $t = (?v, ?v, ?v)$, then one obtains the expression

$$\Pi_{G,v} \left[\rho_{G \leftarrow \mathtt{gid}, v \leftarrow \mathtt{sub}}\left(\sigma_{\mathtt{sub}=\mathtt{pred} \wedge \mathtt{sub}=\mathtt{obj} \wedge \mathtt{pred}=\mathtt{obj}}(\mathtt{Quads})\right)\right]$$

Note that one of the equalities in the selection condition is redundant. As before, for each graph, we get a tuple for each possible value of v.

2 vars: There are two variant of patterns here $t = (?v1, ?v2, t_3)$ and $t = (?v1, ?v2, ?v2)$ and permutations. For the first pattern we get

$$\Pi_{G,v1,v2}\left[\rho_{G\leftarrow\texttt{gid},v1\leftarrow\texttt{sub},v2\leftarrow\texttt{pred}}\left(\sigma_{\texttt{obj}=t_3}(\texttt{Quads})\right)\right]$$

while for the second we get

$$\Pi_{G,v1,v2}\left[\rho_{G\leftarrow\texttt{gid},v1\leftarrow\texttt{sub},v2\leftarrow\texttt{pred}}\left(\sigma_{\texttt{pred}=\texttt{obj}}(\texttt{Quads})\right)\right]$$

It is easy to see that each solution (for each graph) corresponds exactly to one tuple in the evaluation of the translated relation.

3 vars: This case is immediate and being the triple pattern $t = (?v1, ?v2, ?v3)$. The translation is $\Pi_{G,v_1,v_2,v_3}\left[\rho_{G\leftarrow\texttt{gid},v1\leftarrow\texttt{sub},v2\leftarrow\texttt{pred},v3\leftarrow\texttt{obj}}(\texttt{Quads})\right]$, obtaining a relation which is isomorphic to `Quads`, as expected. For each graph i, we obtain a tuple with value i in attribute G and remaining attributes corresponding exactly to one triple in graph i.

`UNION` *pattern* The translated relational algebra expression $[(P_1 \texttt{ UNION } P_2)]^G_{\mathcal{R}}$ is:

$$\begin{array}{c}\Pi_{G,var(P_1)\cup\{v\leftarrow\texttt{unb}\mid v\in var(P_2)\setminus var(P_1)\}}\left([P_1]^G_{\mathcal{R}}\right)\\ \bigcup \\ \Pi_{G,var(P_2)\cup\{v\leftarrow\texttt{unb}\mid v\in var(P_1)\setminus var(P_2)\}}\left([P_2]^G_{\mathcal{R}}\right)\end{array}$$

The relational algebra expressions makes the union of two projections. Each projection will not remove any in-scope variable, and it is used to extend the columns with unbound values in order to obtain a relation with columns $G \cup var(P_1) \cup var(P_2)$, by making unbound the variables that do not occur in the sub-pattern $[P_1]^G_{\mathcal{R}}$ or $[P_2]^G_{\mathcal{R}}$, respectively. Therefore, each subexpression the projection operator will not remove any in-scope attributes of each sub-pattern $[P_1]^G_{\mathcal{R}}$ or $[P_2]^G_{\mathcal{R}}$, and thus it returns exactly as many as tuples as the number of solutions of each sub-pattern by induction hypothesis. The cardinality of the resulting expression is the sum of solutions of each sub-expression, according to the bag semantics of relational algebra.

`AND pattern` The translated relational algebra expression $[(P_1 \texttt{ AND } P_2)]^G_{\mathcal{R}}$ is:

$$\Pi_{\substack{G,\\ var(P_1)-var(P_2),\\ var(P_2)-var(P_1),\\ v_1\leftarrow first(v_1',v_1''),\ldots,\\ v_n\leftarrow first(v_n',v_n'')}}\left[\sigma_{comp}\left(\rho_{\substack{v_1'\leftarrow v_1\\ \vdots\\ v_n'\leftarrow v_n}}\left([P_1]^G_{\mathcal{R}}\right)\bowtie\rho_{\substack{v_1''\leftarrow v_1\\ \vdots\\ v_n''\leftarrow v_n}}\left([P_2]^G_{\mathcal{R}}\right)\right)\right]$$

where *comp* is a conjunction of conditions $v_i' = \texttt{unb} \vee v_i'' = \texttt{unb} \vee v_i' = v_i''$ for each common variable $v_i (1 \leq i \leq n)$. The function $first$ returns the first argument which is not `unb`, or `unb` if both arguments are `unb`.

The joined subexpressions inside the selection have only the common attribute G due to the renaming of common variables. So, every tuple of the subexpression for $[P_1]^G_{\mathcal{R}}$ will join with every tuple of $[P_2]^G_{\mathcal{R}}$, for each graph. If there is a solution with cardinality c_1 of P_1 and solution with cardinality c_2 of P_2, one will obtain $c_1 \times c_2$ tuples in the result of the joined expression. From these possibilities, the selection expression keeps he combinations of solutions which are compatible: for each pair v'_i and v''_i the condition guarantees that at least 1 variable is unbound or have the same value. So, we only keep the tuples for each possible merge (with cardinality $c_1 \times c_2$ since selection keeps duplicate tuples). The projection is necessary to obtain the relation on the original variables, besides G, obtaining the value from the first bound variable, if any. It is also necessary to recall that the bag semantics for the projection operator will obtain as many tuples as the contributing tuples, summing over the obtained solutions as required by the cardinality condition of the `AND` operator (see the definition of the $\cup$ for K-relations) .

`FILTER pattern` The relational algebra expression $[(P\ \texttt{FILTER}\ R)]^G_{\mathcal{R}}$ is

$$\Pi_{G,var(P)}\left[\sigma_{filter}\left([P]^G_{\mathcal{R}} \bowtie E_1 \bowtie \ldots \bowtie E_m\right)\right]$$

where $filter$ is a condition obtained from R where each occurrence of $\texttt{EXISTS}(P_i)$ (resp. $\texttt{NOT EXISTS}(P_i)$) in R is substituted by condition $ex_i <> 0$ (resp. $ex_i = 0$), where ex_i is a new attribute name. Expression $E_i (1 \leq i \leq m)$ is:

$$\Pi_{G,var(P),ex_i\leftarrow 0}\left[\delta(P') - \Pi_{G,var(P)}\left(\sigma_{subst}\left(P' \bowtie \rho_{\substack{v'_1 \leftarrow v_1\\ \vdots \\ v'_n \leftarrow v_n}}(P'_i)\right)\right)\right]$$

$$\cup$$

$$\Pi_{G,var(P),ex_i\leftarrow 1}\left[\delta(P') - \left[\delta(P') - \Pi_{G,var(P)}\left(\sigma_{subst}\left(P' \bowtie \rho_{\substack{v'_1 \leftarrow v_1\\ \vdots \\ v'_n \leftarrow v_n}}(P'_i)\right)\right)\right]\right]$$

where $P' = [P]^G_{\mathcal{R}}$, $P'_i = [P_i]^G_{\mathcal{R}}$, and $subst$ is the conjunction of conditions $v_i = v'_i \vee v_i = \texttt{unb}$ for each variable v_i in $var(P) \cap var(P_i) = \{v_1, \ldots, v_n\}$.

The translation is complex due to the `EXISTS` expressions. Note that each E_i expression returns exactly one tuple for each solution of pattern `P`. Thus, the duplicate removal operations are there just to guarantee this and do no affect the cardinality of the solutions, obtaining one solution for each solution of `P` that obeys to the filter condition. The rest of the translation is more or less immediate, where the condition *subst* does not correspond exactly to the compatibility condition used before, since according to the SPARQL semantics

the variables of pattern $\mathtt{P_1}$ are substituted in the `EXISTS` pattern (we discard the cases where a variable is bound by $\mathtt{P_1}$ and not bound in $\mathtt{P_i}$).

`MINUS pattern` The relational algebra expression $[(P_1 \texttt{ MINUS } P_2)]^G_{\mathcal{R}}$ is

$$[P_1]^G_{\mathcal{R}} \bowtie \left[\delta\left([P_1]^G_{\mathcal{R}}\right) - \Pi_{G,var(P_1)}\left[\sigma_{comp \wedge \neg disj}\left([P_1]^G_{\mathcal{R}} \bowtie \rho_{\substack{v'_1 \leftarrow v_1 \\ \vdots \\ v'_n \leftarrow v_n}}\left([P_2]^G_{\mathcal{R}}\right)\right)\right]\right]$$

where *comp* is a conjunction of conditions $v_i = \texttt{unb} \vee v'_i = \texttt{unb} \vee v_i = v'_i$ for each variable common $v_i (1 \leq i \leq n)$, and $disj$ is the conjunction of conditions $v_i = \texttt{unb} \vee v'_i = \texttt{unb}$ for each variable $v_i (1 \leq i \leq n)$.

The difference expression will return either a tuple of $[P_1]^G_{\mathcal{R}}$ or not, without duplicates. The difference expression evaluation will return solutions of P_1 that do not belong to the expression in the right-hand side of the difference. Since the right-hand side of expression obtains the tuples corresponding to the solutions of $\mathtt{P_1}$ for which there is at least one solution in $\mathtt{P_2}$ that is compatible and not disjoint (no bound variable in common), then we obtain as result the tuples corresponding to solutions of P_1 such that for all solution P_2 the solutions are incompatible or are disjoint (the semantics of `MINUS`). Now, the difference expression will have no duplicates, and thus we keep only the solutions of P_1 that join (i.e. that obey to the condition), without increasing the cardinality of the result.

`OPTIONAL pattern` The relational algebra expression $[(P_1 \texttt{ OPTIONAL } (P_2 \texttt{ FILTER } R))]^G_{\mathcal{R}}$ is

$$\begin{gathered}
[(P_1 \texttt{ AND } P_2)]^G_{\mathcal{R}} \\
\bigcup \\
\Pi_{G,var(P_1) \cup \{v \leftarrow \texttt{unb} \mid v \in var(P_2) \setminus var(P_1)\}} \\
\left[[P_1]^G_{\mathcal{R}} \bowtie \left(\begin{array}{c} \delta\left([P_1]^G_{\mathcal{R}}\right) \\ - \\ \Pi_{G,var(P_1)}\left([(P_1 \texttt{ AND } P_2) \texttt{ FILTER } R]^G_{\mathcal{R}}\right) \end{array}\right)\right]
\end{gathered}$$

According to the semantics of SPARQL 1.1, the `OPTIONAL` pattern evaluation is performed by two operators: a join and a left join. This is particularly clear in the translation, where the join is the first expression and the left join the lower (big) expression below the union operator. The rationale of the translation of the left join operator is identical to the translation of the `MINUS` pattern, except now that we need to make unbound the variables in $\mathtt{P_2}$ but not in $\mathtt{P_1}$. Again, the number of obtained tuples is according to the semantics of SPARQL 1.1: it is the sum of the tuples of the join with the sum of tuples of the left join (guaranteed by the bag semantics of $\cup$).

`GRAPH pattern` The translation of ($\texttt{GRAPH}$ $term$ P_1) has two cases:

– If $term$ is an IRI then $[(\texttt{GRAPH}\ term\ P_1)]^G_{\mathcal{R}}$ is

$$[()]^G_{\mathcal{R}} \bowtie \Pi_{var(P_1)} \left[\Pi_{G'} \left(\rho_{G' \leftarrow \texttt{gid}} \left(\sigma_{term=\texttt{IRI}}(\texttt{Graphs})\right)\right) \bowtie [P_1]^{G'}_{\mathcal{R}}\right]$$

The selection expression obtains a single tuple containing the identifier corresponding to the $term$, or obtains the empty relation if there is no named graph with that IRI. This tuple joins with $[P_1]^{G'}_{\mathcal{R}}$ to limit the results to the intended graph. Moreover it is important that the use of a renamed attribute for the graph is necessary in order to avoid clashes of attributes corresponding to the active graph. Moreover, the graph pattern will return the same results independently of the active graph and this is captured by the join with the empty graph pattern.

– If $term$ is a variable v then $[(\texttt{GRAPH}\ term\ P_1)]^G_{\mathcal{R}}$ is

$$[()]^G_{\mathcal{R}} \bowtie \Pi_{\{v\} \cup var(P_1)} \left[\rho_{G' \leftarrow \texttt{gid}, v \leftarrow \texttt{IRI}} \left(\sigma_{\texttt{gid}>0}(\texttt{Graphs})\right) \bowtie [P_1]^{G'}_{\mathcal{R}}\right]$$

This case is a little more complex, because now we consider all the named graphs ($\texttt{gid} > 0$, and bind the variable v with the corresponding IRIs. The rationale of the construction is the same of the previous case.

To conclude the proof, we just need to analyse the results of the expression corresponding to the full query $[SPARQL(P, D(G), V)]_{\mathcal{R}} = \Pi_V \left[\sigma_{G'=0} \left([()]^{G'}_{\mathcal{R}} \bowtie [P]^{G'}_{\mathcal{R}}\right)\right]$ with respect to the base relations `Graphs` and `Quads`, where G' is a new attribute name and $V \subseteq var(P)$. The selection starts the evaluation at the default graph ($G' = 0$) and projects in the selected variables. The correctness of the translation is now immediate due to the previous induction.

References

1. SPARQL 1.1 query language, 2012. W3C Working Draft 05 January 2012. Available at `http://www.w3.org/TR/2012/WD-sparql11-query-20120105/`.
2. K. Amer. Equationally complete classes of commutative monoids with monus. *Algebra Universalis*, 18:129–131, 1984.
3. A. P. Antoine Zimmermann, Nuno Lopes and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Journal of Web Semantics*, 11:72–95, March 2012.
4. P. Buneman and E. V. Kostylev. Annotation algebras for RDFS. In *Proc. of the 2nd Int. Ws. on the role of Semantic Web in Provenance Management (SWPM 2010)*. CEUR Workshop Proceedings, 2010.
5. R. Dividino, S. Sizov, S. Staab, and B. Schueler. Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *Web Semant.*, 7(3):204–219, 2009.
6. B. Elliott, E. Cheng, C. Thomas-Ogbuji, and Z. M. Ozsoyoglu. A complete translation from SPARQL into efficient SQL. In *Proc. of the 2009 Int. Database Engineering & Applications Symposium*, IDEAS '09, pages 31–42. ACM, 2009.
7. G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring RDF triples to capture provenance. In *Proc. of the 8th Int. Semantic Web Conf.*, ISWC '09, pages 196–212, Berlin, Heidelberg, 2009. Springer-Verlag.
8. F. Geerts and A. Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2):173–185, 2010.
9. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. of PODS'07*, pages 31–40, New York, NY, USA, 2007. ACM.
10. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.
11. A. Polleres. From SPARQL to rules (and back). In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *Proc. of the 16th Int. Conf. on World Wide Web, WWW 2007*, pages 787–796. ACM, 2007.
12. Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1):31–39, 2011.